**TP Architecture des ordinateurs**

**8 TP4 : le corrigé**

**Le fichier Exch.c**

**\_exch: void exch( int n1, int n2 ) {**

cs:02C2 55 push bp

cs:02C3 8BEC mov bp,sp

cs:02C5 83EC02 sub sp,0002

**#EXCH#10: tmp = n1;**

cs:02C8 8B4604 mov ax,[bp+04]

cs:02CB 8946FE mov [bp-02],ax

**#EXCH#11: n1 = n2;**

cs:02CE 8B4606 mov ax,[bp+06]

cs:02D1 894604 mov [bp+04],ax

**#EXCH#12: n2 = tmp;**

cs:02D4 8B46FE mov ax,[bp-02]

cs:02D7 894606 mov [bp+06],ax

**#EXCH#15: }**

cs:02DA 8BE5 mov sp,bp

cs:02DC 5D pop bp

cs:02DD C3 ret

**\_main: clrscr();**

cs:02DE E8F30E call \_clrscr

**#EXCH#22: scanf( "%d", &nombre1 );**

cs:02E1 B8FA04 mov ax,04FA

...

**#EXCH#23: scanf( "%d", &nombre2 );**

cs:02EF B8F804 mov ax,04F8

...

**#EXCH#25: exch( nombre1, nombre2 );**

cs:02FD FF36F804 push word ptr [\_nombre2]

cs:0301 FF36FA04 push word ptr [\_nombre1]

cs:0305 E8BAFF call \_exch

cs:0308 83C404 add sp,0004

**#EXCH#26: printf( "Après\nnombre1 = %d\nnombre2 = %d\n", nom**

cs:030B FF36F804 push word ptr [\_nombre2]

...

**#EXCH#28: return 0;**

cs:031D 33C0 xor ax,ax

**#EXCH#29: }**

cs:031F C3 ret

**Travail à réaliser**

Constatez l'inefficacité de cette procédure à échanger les valeurs de nombre1 et nombre2 en observant le déroulement de exch sous Turbo-Debugger.

* Relevez sur papier l'évolution de la pile et des registres. Faites apparaître le pointeur de pile SP et le pointeur de base de pile BP.

**On s'intéresse d'abord au programme appelant :**

|  |  |
| --- | --- |
| Au départ, la pile est dans un certain état, les informations déjà en pile ne nous intéressent pas ; on les représente sous forme de hachures. | Tp4_01.gif |
| On empile la valeur contenue dans nombre2 :  #EXCH#25: exch( nombre1, nombre2 );  cs:02FD FF36F804 push [\_nombre2] | Tp4_02.gif |
| On empile la valeur contenue dans nombre1 :  cs:0301 FF36FA04 push [\_nombre1] | Tp4_03.gif |
| Puis on exécute l'appel au sous-programme exch :  cs:0305 E8BAFF call \_exch  cs:**0308** 83C404 add sp,0004 | Tp4_04.gif |

IP pointe sur la première instruction de **exch**. Le "**PUSH BP**" n'est pas encore exécuté.

**On s'intéresse maintenant au programme appelé :**

|  |  |
| --- | --- |
| \_exch: void exch( int n1, int n2 ) {  cs:02C2 55 push bp  cs:02C3 8BEC mov bp,sp | Tp4_05.gif |
| cs:02C5 83EC02 sub sp,0002 | Tp4_06.gif |
| #EXCH#10: tmp = n1;  cs:02C8 8B4604 mov ax,[bp+04]  cs:02CB 8946FE mov [bp-02],ax | Tp4_07.gif |
| #EXCH#11: n1 = n2;  cs:02CE 8B4606 mov ax,[bp+06]  cs:02D1 894604 mov [bp+04],ax | Tp4_08.gif |
| #EXCH#12: n2 = tmp;  cs:02D4 8B46FE mov ax,[bp-02]  cs:02D7 894606 mov [bp+06],ax | Tp4_09.gif |
| #EXCH#15: }  cs:02DA 8BE5 mov sp,bp | Tp4_10.gif |
| Avec le "POP BP", BP retrouve son ancienne valeur. Par exemple, on peut supposer qu'il désignait une autre case dans la pile, ce qu'on symbolise par une flèche avec un point d'interrogation.  cs:02DC 5D pop bp | Tp4_11.gif |
| Le RET effectue le retour de sous-programme :  cs:02DD C3 ret | Tp4_12.gif |
| IP pointe l'instruction juste après le CALL dans le programme appelant. Le add sp,4nettoie la pile de 4 octets. La pile a retrouvé son état initial :  cs:0305 E8BAFF call \_exch  cs:0308>83C404 add sp,0004 | Tp4_13.gif |

* Pourquoi les valeurs de nombre1 et nombre2 ne sont-elles pas échangées ? ***parce qu'on ne manipule pas*** **nombre1** ***et*** **nombre2** ***mais des copies de ces variables***
* Qu'est-ce qui est échangé en réalité ? ***les valeurs des copies (*n1** ***et*** **n2*) sont échangées sur la pile***
* Comment modifier le programme pour que les valeurs de nombre1 et nombre2 soient bien échangées ? ***Dans l'appelé, il faut que*** **n1** ***et*** **n2** ***soient des pointeurs d'entiers. Dans l'appelant, il faut passer l'adresse de*** **nombre1** ***et*** **nombre2** ***en argument***

**Travail sur Exch1.c**

*Passage des arguments par adresse.*

**\_exch: void exch( int far \*n1, int far \*n2 ) {**

cs:02C2 55 push bp

cs:02C3 8BEC mov bp,sp

cs:02C5 83EC02 sub sp,0002

**#EXCH1#13: tmp = \*n1;**

cs:02C8 C45E04 les bx,[bp+04]

cs:02CB 268B07 mov ax,es:[bx]

cs:02CE 8946FE mov [bp-02],ax

**#EXCH1#14: \*n1 = \*n2;**

cs:02D1 C45E08 les bx,[bp+08]

cs:02D4 268B07 mov ax,es:[bx]

cs:02D7 C45E04 les bx,[bp+04]

cs:02DA 268907 mov es:[bx],ax

**#EXCH1#15: \*n2 = tmp;**

cs:02DD C45E08 les bx,[bp+08]

cs:02E0 8B46FE mov ax,[bp-02]

cs:02E3 268907 mov es:[bx],ax

**#EXCH1#18: }**

cs:02E6 8BE5 mov sp,bp

cs:02E8 5D pop bp

cs:02E9 C3 ret

**\_main: clrscr();**

cs:02EA E81B0F call \_clrscr

**#EXCH1#24: printf( "Entrez un entier : " ); scanf( "%d", &nombre1 );**

cs:02ED B8AA00 mov ax,00AA

...

**#EXCH1#25: printf( "Puis un autre entier : " ); scanf( "%d", &nombre2 );**

cs:0305 B8C100 mov ax,00C1

...

**#EXCH1#26: printf( "Avant\nnombre1 = %d\nnombre2 = %d\n", nombre1, nombre2 );**

cs:031D FF364405 push word ptr [\_nombre2]

...

**#EXCH1#27: exch( &nombre1, &nombre2 );**

cs:032F 1E push ds

cs:0330 B84405 mov ax,0544

cs:0333 50 push ax

cs:0334 1E push ds

cs:0335 B84605 mov ax,0546

cs:0338 50 push ax

cs:0339 E886FF call \_exch

cs:033C 83C408 add sp,0008

**#EXCH1#28: printf( "Après\nnombre1 = %d\nnombre2 = %d\n", no**

cs:033F FF364405 push word ptr [\_nombre2]

...

**#EXCH1#30: return 0;**

cs:0351 33C0 xor ax,ax

**#EXCH1#31: }**

cs:0353 C3 ret

**Travail à réaliser**

Cette procédure échange bien nombre1 et nombre2.

* Relevez sur papier l'évolution de la pile et des registres. Faites apparaître le pointeur de pile SP et le pointeur de base de pile BP.

**On s'intéresse d'abord au programme appelant :**

Au départ, la pile est dans un certain état, les informations déjà en pile ne nousintéressent pas ; on les représente sous forme de hachures. Considérons d'abord le programme appelant. On suppose que le segment de données contenant nombre1 et nombre2 commence à l'adresse6ABD : 0. Pour que tout fonctionne bien, DS doit contenir  6ABD; c'est donc ce que l'on suppose. On suppose également que les parties offset des adresses de nombre1 et nombre2 sont respectivement 0546 et 0544. Enrésumé, on peut dire que l'adresse complète de nombre1 est 6ABD : 0546 et que l'adresse complète de nombre2 est 6ABD : 0544.

|  |  |
| --- | --- |
|  | Tp4_14.gif |
| #EXCH1#27: exch( &nombre1, &nombre2 );  cs:032F 1E push ds  cs:0330 B84405 mov ax,0544  cs:0333 50 push ax | Tp4_15.gif |
| cs:0334 1E push ds  cs:0335 B84605 mov ax,0546  cs:0338 50 push ax | Tp4_16.gif |
| cs:0339 E886FF call \_exch  cs:**033C** 83C408 add sp,0008 | Tp4_17.gif |

**On s'intéresse maintenant au programme appelé :**

|  |  |
| --- | --- |
| \_exch: void exch( int far \*n1, int far \*n2 ) {  cs:02C2 55 push bp  cs:02C3 8BEC mov bp,sp | Tp4_18.gif |
| cs:02C5 83EC02 sub sp,0002 | Tp4_19.gif |
| #EXCH1#13: tmp = \*n1;  cs:02C8 C45E04 les bx,[bp+04]  cs:02CB 268B07 mov ax,es:[bx]  cs:02CE 8946FE mov [bp-02],ax | **Tp4_20.gif**  Tp4_21.gif |
| #EXCH1#14: \*n1 = \*n2;  cs:02D1 C45E08 les bx,[bp+08]  cs:02D4 268B07 mov ax,es:[bx]  cs:02D7 C45E04 les bx,[bp+04]  cs:02DA 268907 mov es:[bx],ax | Tp4_22.gif  Tp4_23.gif |
| #EXCH1#15: \*n2 = tmp;  cs:02DD C45E08 les bx,[bp+08]  cs:02E0 8B46FE mov ax,[bp-02]  cs:02E3 268907 mov es:[bx],ax | Tp4_24.gif  Tp4_25.gif |

A l'issue de ces instructions, les valeurs de nombre1 et nombre2 ont été échangées.

|  |  |
| --- | --- |
| #EXCH1#18: }  cs:02E6 8BE5 mov sp,bp | Tp4_26.gif |
| cs:02E8 5D pop bp | Tp4_27.gif |
| cs:02E9 C3 ret | Tp4_28.gif |

On se retrouve après l'instruction CALL dans le programme appelant :

|  |  |
| --- | --- |
| cs:0339 E886FF call \_exch  cs:033C 83C408 add sp,0008 | Tp4_29.gif |

* Observez le code généré pour différentes options d'optimisation. Dans le Turbo-C, utilisez le menu Options / Compiler / Opimizations. Rubrique Register variables de la boîte de dialogue.
  1. pas d'optimisation, (choix None) ***la variable*** **tmp** ***est sur la pile***
  2. optimisation par le mot-clé register appliqué à tmp, (choix Register keyword) ***la variable*** **tmp** ***est dans le registre*** **SI*,***
  3. optimisation automatique (choix Automatic) ***la variable*** **tmp** ***est sur la pile.***
* Quel couple de registres est utilisé par le compilateur pour les pointeurs far ? ***le couple*** **es:bx**
* Quelle est l'adresse segmentée complète de nombre1 et nombre2 ? ***cela dépend de l'exécution et/ou du compilateur. Ici,*** **6ABD:0546** ***pour*** **nombre1** ***et*** **6ABD:0544** ***pour*nombre2**

**Travail sur Exch2.c**

*Optimisation de la procédure d'échange.*

**Travail à réaliser**

Codez la procédure exch en assembleur de façon plus efficace que le compilateur, c'est-à-dire avec moins d'instructions ou des instructions moins coûteuses.

#include <stdio.h>

void exch( int far \*n1, int far \*n2 ) {

**asm {**

push ds // sauvegarder ds (car on va le modifier et il est utilisé par l'appelant)

**// ax = \*n1**

les di,[n1] // es:di = n1 (n1 occupe 4 octets car c'est un pointeur far)

mov ax,es:[di] // ax = valeur pointée par es:di (c-à-d la valeur de nombre1)

**// \*n1 = \*n2**

lds si,[n2] // ds:di = n2 (n2 occupe 4 octets car c'est un pointeur far)

mov bx,[si] // bx = valeur pointée par ds:si (c-à-d la valeur de nombre2)

mov es:[di],bx // endroit pointé par es:di (c-à-d nombre1) = bx (valeur de nombre2)

**// \*n2 = ax**

mov [si],ax // endroit pointé par ds:di (c-à-d nombre2) = ax (valeur de nombre1)

pop ds // récupérer la valeur de ds de l'appelant

**}**

}

**Christophe Tombelle.**  
**Copyright © 1997- [Telecom Lille].**  
**Revisé: 21-01-2014.**